МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

# Кафедра проектирования информационно-компьютерных систем

Отчет по предмету

Программирование на языке ассемблера

Лабораторная работа №4

**«СОЗДАНИЕ ВИДЕОИГРЫ.»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы 150501  Смоленский Н.О. | **Проверил:**  Туровец Н.О. |

Минск 2022

**Цель работы**: ознакомиться в рамках создания видеоигры с обработкой нажатий кнопок клавиатуры, рассмотреть прямой доступ к видеопамяти с целью формирования игрового поля и информации для пользователя.

**Теоретические сведения:**

1. *Прямой доступ к видеопамяти.*

В большинстве текстовых видеорежимов под видеопамять отводится специальная область памяти, начинающаяся с абсолютного адреса B800h:0000h и заканчивающаяся адресом B800h:FFFFh. Все, что программа запишет в эту область памяти, будет пересылаться в память видеоадаптера и отображаться на экране.

В текстовых режимах для хранения каждого изображенного символа используются два байта:

-- байт с ASCII-кодом символа;

-- байт атрибута символа (указывает цвет символа и фона, мигание).

Байт атрибута символа имеет следующий формат (биты):

-- 7 – символ мигает (по умолчанию) или фон яркого цвета (если его действие было переопределено прерыванием 10h).

-- 6 – 4 – цвет фона.

-- 3 – символ яркого цвета (по умолчанию) или фон мигает (если его действие было переопределено прерыванием 11h).

-- 2 – 0: цвет символа.

Таким образом, по адресу B800h:0000h лежит байт с кодом символа,  
находящимся в верхнем левом углу экрана; по адресу B800h:0001h лежит  
атрибут этого символа; по адресу B800h:0002h лежит код второго символа в верхней строке экрана и т.д.

Для установки требуемого программе видеорежима используется прерывание 10h (видеосервис) BIOS. Видеорежимы отличаются друг от друга разрешением (для графических) и количеством строк и столбцов (для текстовых), а также количеством возможных цветов. В данной лабораторной работе использование графических режимов видеоадаптера не требуется, поэтому в описании прерываний эта информация будет опущена.

-- Прерывание BIOS 10h, функция 00 – установить видеорежим;

-- Прерывание BIOS 10h, функция 01 – установить размер курсора;

-- Прерывание BIOS 10h, функция 02 – установить положение курсор;

-- Прерывание BIOS 10h, функция 03 – получить положение и размер курсора (каждая страница использует собственный независимый курсор);

-- Прерывание BIOS 10h, функция 05 – установить активную страницу (доступно для цветных текстовых режимов);

-- Прерывание BIOS 10h, функция 06 – прокрутка экрана вверх (вставка чистых строк снизу);

-- Прерывание BIOS 10h, функция 07 – прокрутка экрана вниз (вставка чистых строк сверху).

Прерывание 10h также обеспечивает функции вывода данных на уровне BIOS:

-- Прерывание BIOS 10h, функция 08 – считать символ и атрибут символа в текущей позиции курсора;

-- Прерывание BIOS 10h, функция 09 – вывести символ с заданным атрибутом на экран;

-- Прерывание BIOS 10h, функция 0Ah – вывести символ с текущим атрибутом на экран;

-- Прерывание BIOS 10h, функция 0Eh – вывести символ в режиме телетайпа;

-- Прерывание BIOS 10h, функция 13h – вывести строку символов с заданными атрибутами.

2. *Обработка нажатий кнопок клавиатуры.*

По сравнению с функциями DOS, прерывание BIOS 16h предоставляет больше возможностей для считывания данных и управления клавиатурой и такой доступ практически эквивалентен по производительности прямому доступу к буферу клавиатуры.

Каждой клавише на клавиатуре соответствует уникальный код, называемый скан-код. Этот код посылается клавиатурой при каждом нажатии и отпускании клавиши и обрабатывается BIOS – записывается в кольцевой буфер клавиатуры.

Функции прерывания 16h:

-- Функция 00h (10h, 20h) – чтение символа с ожиданием;

-- Функция 01h (11h, 21h) – проверка наличия символа в буфере;

-- Прерывание BIOS 16h, функция 02h (12h, 22h) – получить состояние клавиатуры;

-- Прерывание BIOS 16h, функция 05 – поместить символ в буфер клавиатуры.

К буферу клавиатуры также можно обратиться напрямую – буфер находится по адресу 0000h:041Eh и занимает 16 слов, по 0000h:043Dh включительно. Каждый символ хранится в буфере в виде слова, в таком же виде, как возвращает функция 01h прерывания INT 16h.

3. *Доступ к системным часам.*

Персональный компьютер содержит два устройства для управления процессами:

-- часы реального времени (RTC) – имеют автономное питание, используются для чтения/установки текущих даты и времени, установки будильника и для вызова прерывания IRQ8 (INT 4Ah) каждую миллисекунду;

-- системный таймер – используется одновременно для управления контроллером прямого доступа к памяти, для управления динамиком и как генератор импульсов, вызывающий прерывание IRQ0 (INT 8h) 18,2 раза в секунду.

Для видеоигры, создаваемой в данной лабораторной работе, указанные  
выше устройства лучше всего использовать на уровне функций DOS или BIOS как средство для определения текущего времени, организации задержек и формирования случайных чисел.

Управление часами RTC и внутренними часами операционной системы  
средствами DOS:

-- Функция DOS 2Ah (INT 21h) – считать дату;

-- Функция DOS 2Bh (INT 21h) – установить дату;

-- Функция DOS 2Ch (INT 21h) – считать время;

-- Функция DOS 2Dh (INT 21h) – установить время.

BIOS позволяет управлять часами PTC напрямую:

-- Прерывание BIOS 1Ah, функция 02h – считать время RTC;

-- Прерывание BIOS 1Ah, функция 03h – установить время RTC;

-- Прерывание BIOS 1Ah, функция 04h – считать дату RTC;

-- Прерывание BIOS 1Ah, функция 05h – установить дату RTC;

Для работы со счетчиком времени в BIOS есть функции:

-- Прерывание BIOS 1Ah, функция 00h – прочитать значение счетчика времени;

-- Прерывание BIOS 1Ah, функция 01h – установить значение счетчика времени;

-- Прерывание BIOS 15h, функция 86h – формирование задержки таймера;

-- Прерывание BIOS 15h, функция 83h – управление работой счётчика;

**Код программы:**

*clearScreen MACRO*

*push ax ; save znachenie ax*

*mov ax, 0003h ; 00 - ustanovit videorezhim, clear the screen. 03h - rezhim 80x25*

*int 10h ; call for prerivaniye to make command*

*pop ax ; vosstanavit' znachenie ax*

*ENDM*

*.model small*

*.stack 100h*

*.data*

*KUpSpeed equ 48h ; Up key*

*KDownSpeed equ 50h ; Down key*

*KMoveUp equ 11h ; W key*

*KMoveDown equ 1Fh ; S key*

*KMoveLeft equ 1Eh ; A key*

*KMoveRight equ 20h ; D key*

*KExit equ 01h ; ESC key*

*xSize equ 80 ; width of console*

*ySize equ 25 ; height of console*

*xField equ 50 ; width of field*

*yField equ 21 ; height of field*

*oneMemoBlock equ 2 ; size of one kletka of console*

*scoreSize equ 4 ; size of score table*

*videoStart dw 0B800h ; smeschenie videobufera*

*dataStart dw 0000h ; byte with code of symbol of left high corner*

*space equ 0020h ; emoty block with black background*

*snakeBodySymbol equ 0A6Fh ; symbol of snakes body*

*appleSymbol equ 040Fh ; symbol of apple*

*VWallSymbol equ 0FBAh ; symbol of vertical wall*

*HWallSymbol equ 0FCDh ; symbol of horizontal wall*

*BWallSymbol equ 4020h ;*

*VWallSpecialSymbol equ 0FCCh ; symbol of peresecheniye walls*

*fieldSpacingBad equ space, VWallSymbol, xField dup(space)*

*fieldSpacing equ fieldSpacingBad, VWallSymbol*

*Spc equ 0F20h ; probel with red background and white sumbol colour*

*screen dw xSize dup(space)*

*dw space, 0FC9h, xField dup(HWallSymbol), 0FCBh, xSize - xField - 5 dup(HWallSymbol), 0FBBh, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F53h, 0F63h, 0F6Fh, 0F72h, 0F65h, 0F3Ah, Spc*

*score dw scoreSize dup(0F30h), xSize - xField - scoreSize - 13 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F53h, 0F70h, 2 dup(0F65h), 0F64h, 0F3Ah, Spc*

*speed dw 0F31h, 16 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F43h, 0F6Fh, 0F6Eh, 0F74h,0F72h, 0F6Fh, 0F6Ch,0F73h, 0F3Ah, 15 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F57h, Spc, 0FC4h, Spc, 0F55h, 0F70h, 18 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F53h, Spc, 0FC4h, Spc, 0F44h, 0F6Fh, 0F77h ,0F6Eh, 16 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F41h, Spc, 0FC4h, Spc, 0F4Ch, 0F65h, 0F66h ,0F74h, 16 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F44h, Spc, 0FC4h, Spc, 0F52h, 0F69h, 0F67h ,0F68h, 0F74h, 15 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, Spc, 0F45h, 0F73h,0F63h, Spc, 0FC4h, Spc, 0F45h, 0F78h, 0F69h ,0F74h, xSize - xField - 16 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw fieldSpacing, xSize - xField - 5 dup(Spc), VWallSymbol, space*

*dw space, 0FC8h, xField dup(HWallSymbol), 0FCAh, xSize - xField - 5 dup(HWallSymbol), 0FBCh, space*

*dw xSize dup(space)*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; BANNER*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*widthOfBanner equ 40 ;*

*allWidth equ 80 ;*

*black equ 0020h ;*

*white equ 4020h ;*

*blackVWallSymbol equ 00FBAh*

*blackHWallSymbol equ 00FCDh*

*wastedBanner dw 00FC9h, widthOfBanner-2 dup(blackHWallSymbol), 0FBBh*

*dw blackVWallSymbol, widthOfBanner-2 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, 4 dup(black), white, 5 dup(black), white, 2 dup(black), 2 dup(white), black, 4 dup(white), black, 3 dup(white), black, 3 dup(white), black, 3 dup(white), 6 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, 4 dup(black), white, 5 dup(black), white, black, white, black, white, black,white, black, black, black, black, black, white, 2 dup(black), white, 2 dup(black), black, white, black, black, white, 5 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, 5 dup(black), 3 dup(white, black), black, 3 dup(white), black, 4 dup(white), 2 dup(black), white, 2 dup(black), 2 dup(white), 2 dup(black), white, 2 dup(black), white, 5 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, 5 dup(black), 3 dup(white, black), black, white, black, white, 4 dup(black), white, 2 dup(black), white, 2 dup(black), white, 2 dup(black), black, white, 2 dup(black), white, 5 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, 6 dup(black), 2 dup(white, black), 2 dup(black), white, black, white, black, 4 dup(white), 2 dup(black), white, 2 dup(black), 3 dup(white), black, 3 dup(white), 6 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, widthOfBanner-2 dup(black), blackVWallSymbol*

*dw blackVWallSymbol, 7 dup(black) ,08F50h, 08F72h, 08F65h, 08F73h, 08F73h, 08F00h, 08F61h, 08F6Eh, 08F79h, 08F00h, 08F6Bh, 08F65h, 08F79h, 08F00h, 08F74h, 08F6Fh, 08F00h, 08F65h, 08F78h, 08F69h, 08F74h, 10 dup(black), blackVWallSymbol*

*dw 0FC8h, widthOfBanner-2 dup(blackHWallSymbol), 0FBCh*

*snakeMaxSize equ 30*

*snakeSize db 3*

*PointSize equ 2*

*snakeBody dw 1D0Dh, 1C0Dh, 1B0Dh, snakeMaxSize-2 dup(0000h)*

*stopVal equ 00h*

*forwardVal equ 01h*

*backwardVal equ -1*

*Bmoveright db 01h*

*Bmovedown db 00h*

*minWaitTime equ 1*

*maxWaitTime equ 9*

*waitTime dw maxWaitTime*

*deltaTime equ 1*

*.code*

*main:*

*mov ax, @data*

*mov ds, ax*

*mov dataStart, ax ; upload data*

*mov ax, videoStart ; put in ax beggining of output in videobuffer*

*mov es, ax ; move ax to es-segment register*

*xor ax, ax*

*clearScreen ; clear console*

*call initAllScreen ; initialize screen*

*call mainGame ; go to main cycle of game*

*to\_close:*

*call printBanner*

*mov ah,7h ; 7h - console input w/o eho(wait for input to close app)*

*int 21h*

*esc\_exit:*

*clearScreen ; clear console*

*mov ah, 4ch ; 4ch- function to exit*

*int 21h*

*;ZF = 1 - buffer is empty*

*;AH = scan-code*

*CheckBuffer MACRO ; check if there was input*

*mov ah, 01h ; 01h- read symbol with waiting and check for ctrl-break*

*int 16h*

*ENDM*

*ReadFromBuffer MACRO ; read what was inputed*

*mov ah, 00h ; 00h - wait for next input*

*int 16h*

*ENDM*

*;result in cx:dx*

*GetTimerValue MACRO*

*push ax ; save what is in ax*

*mov ax, 00h ; get time*

*int 1Ah ; preryvaniye for time*

*pop ax ; vosstanovlenye what is in ax*

*ENDM*

*printBanner PROC*

*push es*

*push 0B800h*

*; 0b800h*

*pop es ; ES=0B800h*

*;*

*mov di, 7\*allWidth\*2 + (allWidth - widthOfBanner)*

*mov si, offset wastedBanner*

*mov cx, 10*

*cld ; DF =0*

*loopPrintBanner:*

*push cx*

*mov cx, widthOfBanner*

*rep movsw ; movsw - what is in DS:(E)SI to ES:(E)D*

*add di, 2\*(allWidth - widthOfBanner)*

*pop cx*

*loop loopPrintBanner*

*std ;DF=1*

*pop es*

*ret*

*ENDP*

*initAllScreen PROC*

*mov si, offset screen ; put to di*

*xor di, di*

*; now ds:si point on symbols we want to input*

*; aand es:di on di'yi sembol of console*

*mov cx, xSize\*ySize ; put to dz amount of symbols in console*

*rep movsw ; rewrite all cx symbols from ds:si to console es:di*

*xor ch, ch*

*mov cl, snakeSize ; put to cl size of snake*

*mov si, offset snakeBody; to si upload smescheniye of begining of snake*

*loopInitSnake: ; cycle to uotput snakes body*

*mov bx, [si] ; put to si ocherednoy symbol of snake's body*

*add si, PointSize ; add to si PointSize (2), because every point is 2 bytes (colour+symbol)*

*call CalcOffsetByPoint ; get smeschenie of outputed symbol*

*mov di, bx ; put in di position*

*mov ax, snakeBodySymbol ; put to ax outputed symbol*

*stosw ; output stosw - save es:di with what is in al or ax*

*loop loopInitSnake*

*call GenerateRandomApple; generate random apple*

*ret*

*ENDP*

*;get smescheniye of videobuffer like (bh + (bl \* xSize))\*oneMemoBlock*

*;input: coordinates (x,y) in bx*

*;output: smescheniye in bx*

*CalcOffsetByPoint PROC*

*push ax ; save what is in ax and dx*

*push dx*

*xor ah, ah*

*mov al, bl*

*mov dl, xSize ; xSize - size of string*

*mul dl ; al\*dl*

*mov dl, bh*

*xor dh, dh*

*add ax, dx ; ax+dx*

*mov dx, oneMemoBlock ; oneMemoBlock - length of every block*

*mul dx*

*mov bx, ax*

*pop dx ; vosstanovleniye ax and dx*

*pop ax*

*ret*

*ENDP*

*;move body of snale in massive*

*;delete last element*

*;colour last element*

*MoveSnake PROC*

*push ax*

*push bx*

*push cx*

*push si ; save what is in registers*

*push di*

*push es*

*mov al, snakeSize*

*xor ah, ah*

*mov cx, ax*

*mov bx, PointSize ; put size of one dot to bx*

*mul bx ; now ax-real position in memory otnositelno nachala massiva*

*mov di, offset snakeBody; put to di smescheniye of head af snake*

*add di, ax ; di - adress of next element after last element of massiv*

*mov si, di*

*sub si, PointSize ; si-adress of last massiv element*

*push di ; save what is in di*

*; delete end of snake from screen*

*mov es, videoStart ; put to es smeschenie videobufera*

*mov bx, ds:[si] ; put to bx last element of snake*

*call CalcOffsetByPoint ; calculate its position on screen*

*mov di, bx ; put to di position we need to clear*

*mov ax, space ; put empty point to ax*

*stosw ; write(put what is in ax to es:di)*

*pop di ; vosstanavlivayem di*

*mov es, dataStart ; to work with data(before es pointed to videobufer)*

*std ; df=1; go from end to start*

*rep movsw ; rewrite frfom ds:si to es:di (si - prelast elem of snake, di - last element)*

*;smeschaem all snake on 1 step*

*mov bx, snakeBody ; put to dx head of snake*

*add bh, Bmoveright ; renew head coordinates*

*add bl, Bmovedown*

*mov snakeBody, bx ; save new head*

*;now all body is moved ;*

*pop es*

*pop di*

*pop si*

*pop cx ; vosstanavlivayem registers*

*pop bx*

*pop ax*

*ret*

*ENDP*

*mainGame PROC*

*push ax*

*push bx*

*push cx*

*push dx ; save what is in registers*

*push ds*

*push es*

*checkAndMoveLoop:*

*CheckBuffer ; check if there was input*

*jnz skipJmp2 ; if yes - skipJmp2*

*jmp far ptr noSymbolInBuff ; if no- noSymbolInBuff*

*skipJmp2:*

*ReadFromBuffer ; read symbol for buffer*

*cmp ah, KExit ; if exit was pressed*

*jne skipJmp ; else skipJmp*

*jmp far ptr esc\_exit ; go to endf game endLoop*

*skipJmp:*

*cmp ah, KMoveLeft ; if "left" was pressed*

*je setMoveLeft*

*cmp ah, KMoveRight ; if "right" was pressed*

*je setMoveRight*

*cmp ah, KMoveUp ;if "up" was pressed*

*je setMoveUp*

*cmp ah, KMoveDown ;if "down" was pressed*

*je setMoveDown*

*cmp ah, KUpSpeed ; move up key is pressed*

*je setSpeedUp*

*cmp ah, KDownSpeed ; move down key is pressed*

*je setSpeedDown*

*jmp noSymbolInBuff*

*setMoveLeft:*

*mov al, Bmoveright ; check if tried to switch directions*

*cmp al, forwardVal*

*jne setMoveLeft\_ok*

*jmp noSymbolInBuff*

*setMoveLeft\_ok:*

*mov Bmoveright, backwardVal ; direction right - negative*

*mov Bmovedown, stopVal ; direction down - null*

*jmp noSymbolInBuff*

*setMoveRight:*

*mov al, Bmoveright ; check if tried to switch directions*

*cmp al, backwardVal*

*jne setMoveRight\_ok*

*jmp noSymbolInBuff*

*setMoveRight\_ok:*

*mov Bmoveright, forwardVal ; direction right - positive*

*mov Bmovedown, stopVal ; direction down - null*

*jmp noSymbolInBuff*

*setMoveUp:*

*mov al, Bmovedown ;check if switch directions*

*cmp al, forwardVal*

*jne setMoveUp\_ok*

*jmp noSymbolInBuff*

*setMoveUp\_ok:*

*mov Bmoveright, stopVal ;direction down-negative*

*mov Bmovedown, backwardVal*

*jmp noSymbolInBuff*

*setMoveDown:*

*mov al, Bmovedown ;check if switch direction*

*cmp al, backwardVal*

*jne setMoveDown\_ok*

*jmp noSymbolInBuff*

*setMoveDown\_ok:*

*mov Bmoveright, stopVal ;direction right - null*

*mov Bmovedown, forwardVal ;direction down - positive*

*jmp noSymbolInBuff*

*setSpeedUp:*

*mov ax, waitTime ;move delay to ax*

*cmp ax, minWaitTime ;compare it to min*

*je noSymbolInBuff ;if min - let it go*

*sub ax, deltaTime ;make time less*

*mov waitTime, ax ;new delay*

*mov es, videoStart*

*mov di, offset speed - offset screen*

*mov ax, es:[di]*

*inc ax*

*mov es:[di], ax*

*jmp noSymbolInBuff*

*setSpeedDown:*

*mov ax, waitTime*

*cmp ax, maxWaitTime*

*je noSymbolInBuff*

*add ax, deltaTime*

*mov waitTime, ax*

*mov es, videoStart*

*mov di, offset speed - offset screen*

*mov ax, es:[di]*

*dec ax*

*mov es:[di], ax*

*jmp noSymbolInBuff*

*noSymbolInBuff:*

*call MoveSnake ;move snake on screen*

*mov bx, snakeBody ;move to bx head of snake*

*checkSymbolAgain:*

*call CalcOffsetByPoint ;in bx now- smescheniye yacheiki konsoli with new head*

*mov es, videoStart ;to es smescheniye videobufera*

*mov ax, es:[bx] ;put to ax symbol where snake should be*

*cmp ax, appleSymbol ;if this symbol is apple*

*je AppleIsNext*

*cmp ax, snakeBodySymbol ;if this symbol is body*

*je SnakeIsNext*

*cmp ax, HWallSymbol ;if symbol is horizontal wall*

*je PortalUpDown*

*cmp ax, VWallSymbol ;if symbol is vertical wall*

*je PortalLeftRight*

*cmp ax, BWallSymbol ;if symbol is horizontal wall*

*je SnakeIsNext*

*cmp ax, VWallSpecialSymbol ;if it is cross*

*je PortalLeftRight*

*jmp GoNextIteration*

*AppleIsNext:*

*call incSnake ;make snake longer*

*call GenerateRandomApple ;new apple*

*call incScore ;make score bigger*

*jmp GoNextIteration ;next step*

*SnakeIsNext:*

*jmp endLoop ;end of game*

*PortalUpDown:*

*mov bx, snakeBody ;put head to bx*

*sub bl, yField ;y-height of console*

*cmp bl, 0 ;check if high or low board*

*jg writeNewHeadPos ;redraw head of snake*

*; if upper wall*

*add bl, yField\*2 ; new coordinates*

*writeNewHeadPos:*

*mov snakeBody, bx ; new head*

*jmp checkSymbolAgain ; check again*

*PortalLeftRight:*

*mov bx, snakeBody*

*sub bh, xField*

*cmp bh, 0*

*jg writeNewHeadPos ; same analogy with vertical wall*

*add bh, xField\*2*

*jmp writeNewHeadPos*

*GoNextIteration:*

*mov bx, snakeBody ;bx-new head*

*call CalcOffsetByPoint ;count position*

*mov di, bx ; now in di new smeschniye bx in console*

*mov ax, snakeBodySymbol ;put in ax symbol of snake*

*stosw ;input in console*

*call Sleep ; delay*

*jmp checkAndMoveLoop*

*endLoop:*

*pop es*

*pop ds*

*pop dx ;vosstanavlivayem registers*

*pop cx*

*pop bx*

*pop ax*

*ret*

*ENDP*

*Sleep PROC*

*push ax*

*push bx ; save registers*

*push cx*

*push dx*

*GetTimerValue ; get current time*

*add dx, waitTime ; dx+waittime*

*mov bx, dx ; put it to bx*

*checkTimeLoop:*

*GetTimerValue ;get current time*

*cmp dx, bx ; ax - current value, bx - needed value*

*jl checkTimeLoop ;if early-go to next iteration*

*pop dx*

*pop cx*

*pop bx ;vosstanovleniye registers*

*pop ax*

*ret*

*ENDP*

*GenerateRandomApple PROC*

*push ax*

*push bx*

*push cx ; save registers*

*push dx*

*push es*

*mov ah, 2Ch ; read current time*

*int 21h ; ch - hour, cl - min, dh - sec, dl - msec*

*mov al, dl*

*mul dh ; now to ax random number*

*xor dx, dx*

*mov cx, 04h*

*div cx*

*mov bh, dl*

*toTemplate:*

*push ax*

*mov ax, [si]*

*mov [di],ax*

*pop ax*

*add di, PointSize*

*add si, PointSize*

*loop toTemplate*

*loop\_random:*

*mov ah, 2Ch ; read current time*

*int 21h ; ch - h, cl - min, dh -sec, dl-msec*

*mov al, dl ; get random number*

*mul dh ;now in ax random number*

*xor dx, dx*

*mov cx, xField ;in dx width of field*

*div cx ;get num of string of apple*

*add dx, 2 ;add smescheniye from start of os'*

*mov bh, dl ;save x*

*xor dx, dx*

*mov cx, yField*

*div cx ; analogically for y*

*add dx, 2*

*mov bl, dl ;now in bx - apple coordinates*

*push bx*

*call CalcOffsetByPoint;count smescheniye*

*mov es, videoStart ;to es - start of videobufer*

*mov ax, es:[bx] ;to ax - symbol where apple will be*

*pop bx*

*cmp ax, space ;compare with empty dot*

*jne loop\_random ;if not empty -new make coordinates*

*push bx*

*call CalcOffsetByPoint;count smescheniye*

*mov es, videoStart ;in es start of videobufer*

*mov ax, appleSymbol;*

*mov es:[bx], ax ;output apple symbol*

*pop bx*

*pop es*

*pop dx*

*pop cx ;vosstanavlivayem registers*

*pop bx*

*pop ax*

*ret*

*ENDP*

*;save tail of snake if no overloading*

*incSnake PROC*

*push ax*

*push bx ; save registers*

*push di*

*push es*

*mov al, snakeSize*

*cmp al, snakeMaxSize*

*je return ;if max - exit*

*; make snake longer*

*inc al ; al+1*

*mov snakeSize, al ; new size*

*dec al ; al-1*

*mov bl, PointSize ; vosstanavlivayem end*

*mul bl ;now in ax nuzhnoye smescheniye*

*mov di, offset snakeBody*

*add di, ax ; di now - point of vosstanovlenie*

*mov es, dataStart ;put data to es*

*mov bx, es:[di] ;in bx - point to vosstanovit'*

*call CalcOffsetByPoint;get coordinates*

*mov es, videoStart ;in es-smeschenie videobufera*

*mov es:[bx], snakeBodySymbol ;put one dot of body*

*return:*

*pop es*

*pop di ;vosstanavlivayem registers*

*pop bx*

*pop ax*

*ret*

*ENDP*

*incScore PROC*

*push ax*

*push es*

*push si*

*push di*

*mov es, videoStart*

*mov cx, scoreSize ;max pos value*

*mov di, offset score + (scoreSize - 1)\*oneMemoBlock - offset screen*

*loop\_score:*

*mov ax, es:[di]*

*cmp al, 39h ;'9' symbol*

*jne nineNotNow*

*sub al, 9*

*mov es:[di], ax*

*sub di, oneMemoBlock ;return to symbol back*

*loop loop\_score*

*jmp return\_incScore*

*nineNotNow:*

*inc ax*

*mov es:[di], ax*

*return\_incScore:*

*pop di*

*pop si*

*pop es*

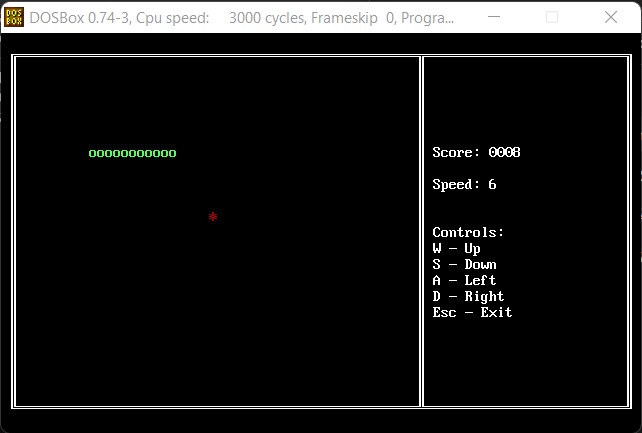
*pop ax*

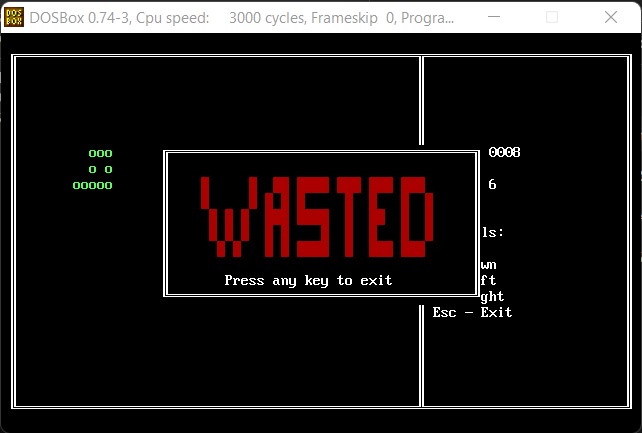
*ret*

*ENDP*

*end main*

**Результат работы:**

****

****

**Вывод**

В ходе выполнения лабораторной работы в рамках создания видеоигры было проведено ознакомление с обработкой нажатий кнопок клавиатуры, рассмотрен прямой доступ к видеопамяти с целью формирования игрового поля и информации для пользователя.